# Resource for Engineering Data in STEP

## Part 50: Mathematical Constructs

**Philip G. Kraushar, Ph. D.**
**The Boeing Company**
**Last revised: Nov 3, 1999**

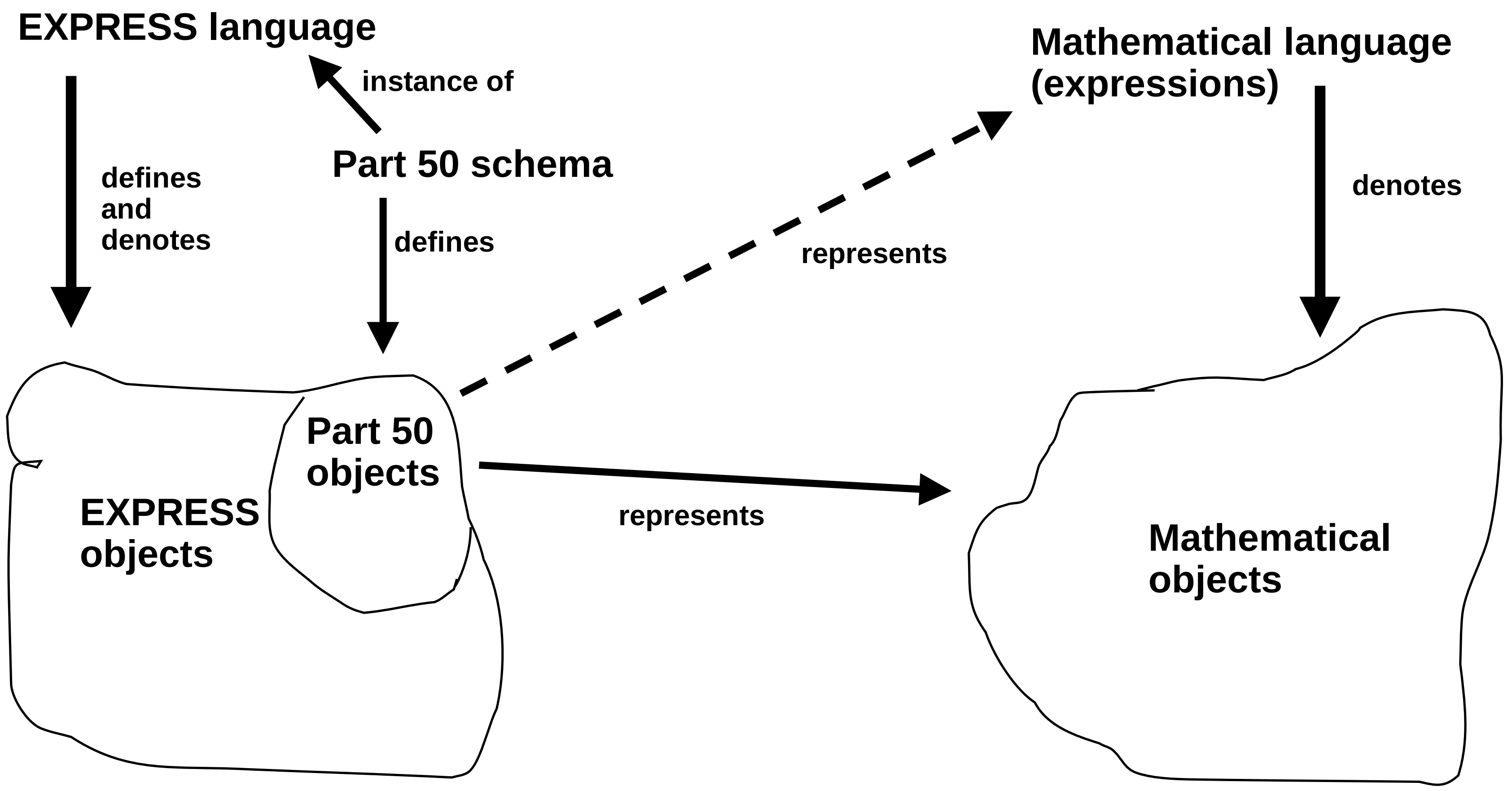
## Scope of Part 50

**“This part of ISO 10303 specifies the resource constructs for the explicit representation of mathematical structures and data related to properties of a product.**

**The following are within the scope of the mathematical functions schema:**
**--- multi-dimensional tables;**
**--- mathematical expressions;**
**--- mathematical functions;**
**--- mathematical spaces.**

**The following are outside the scope of the mathematical functions schema in this part of ISO 10303:**
**--- the context of application;**
**--- the physical units;**
**--- the non-mathematical semantics.”**

# The Big Picture



# Status: Done

- **Clause 7 (mathematical_representation_schema) extracted from Part 42 edition 2 CD and embedded in a new Part 50 (as suggested by the Japanese ballot comment)**
- **Proposed name of Part 50: Mathematical Constructs**
- **Proposed Part 50 schema name: mathematical_functions_schema**
- **Major structural change: All entity types now subtyped from ISO13584-20 (PLIB) expression supertypes (as suggested by discussion based on the French ballot comment)**
- **New draft document completed (218 pages) and on SOLIS as WG12 N442.**
- **All ballot comments that were in scope for this document have been addressed.**

## Status: Not Done

- **EXPRESS compilations**
- **QC revision by author after careful study of SD**

- **Further discussion of ballot comment resolutions and other non-consensus issues by "mathrep" exploder**

**In particular,**

- **Connections from other 40-series Parts**
- **Use by new AP's**
- **Validation?**
- **Extension to cover material in original proposal?**
- **Extension to support functions involving EXPRESS entities (for Parametrics, for example)?**

**(These were out of scope for the DIS document itself.)**

From Lillehammer

## Extensions to generic_expressions_schema

Boeing Phantom Works
Mathematics & Computing Technology

- **Added quantifier_expression as subtype of multiple_arity_generic_expression to deal with logical quantifiers**
- **Added bound_variable_semantics and free_variable_semantics as instantiable subtypes of variable_semantics**
- **Added functions free_variables_of() and is_constant_expression()**
- **Possibly add dependent_variable_definition as a subtype of unary_generic_expression with connections to variable_semantics (discuss in context of later example)**

## Additions caused by integration with PLIB expressions

- Added maths_variable and four special subtypes with dual inheritance from maths_variable and the four instantiable subtypes of ISO13584_expressions_schema.variable, respectively
- Added five elementary subtypes of generic_literal: logical_literal, binary_literal, maths_enum_literal, real_tuple_literal, and integer_tuple_literal
- Added one non-elementary subtype of generic_literal - atom_based_literal - and supporting types: atom_based_value and atom_based_tuple
- Added functions is_maths_expression() and values_space_of()
- Added function_application and elementary_function_application as subtypes of multiple_arity_generic_expression

From Lillehammer

## Entity maths_space Changes

Boeing Phantom Works
Mathematics & Computing Technology

- Subtyped from generic_expression
- All subtypes of maths_space subtyped from generic_literal
- Space (Set) operations (e.g. union, intersection, difference, power set, Cartesian product) not added because not needed and raised too many new issues.
- New subtypes of maths_space:
  - finite_space
  - function_space
  - representable_spaces_space

From Lillehammer

# Entity maths_function Changes (1 of 2)

- Subtyped from generic_expression
- All instantiable subtypes are subtyped from generic_literal, unary_generic_expression or multiple_arity_generic_expression
- Domain and range attributes changed to derived attributes
- Parameters attribute removed (replaced by free_variables_of())
- Added new subtypes (in response to ballot comments):
  - imported_curve_function
  - imported_surface_function
  - imported_volume_function
  - MathML_function
- Replaced subtype compound_function with composed_function, function_application, elementary_function_application, and expression_function

# Entity maths_function Changes (2 of 2)

- Improved names:
  - linear_function ==> homogeneous_linear_function
  - affine_function ==> general_linear_function
  - general_b_spline_function ==> b_spline_function
  - table_function ==> explicit_table_function
  - sparse_matrix ==> basic_sparse_matrix
- Changed:
  - simplest_…_array ==> listed_…_data
  - skew_sym_matrix eliminated (now handled by skew_symmetric attribute of symmetric matrix)
  - symmetric_banded_matrix (now subtype of symmetric_matrix)

# Changes since Lillehammer

- Added dependent_variable_definition as subtype of unary_generic_expression
- Removed enumeration_space, aggregate_space, representable_spaces_space, integer_interval, and real_interval
- Changed product_space and tuple_space into select types
- Removed elementary_function_application
- Added finite_function, expression_denoted_function, and imported_point_function
- Split selection_insertion_function from selection_function
- Inserted abstract supertype application_defined_function above mathml_function as general purpose AP extension mechanism.
- Renamed expression_function to abstracted_expression_function
- Revised list of elementary functions
- Will add composed_function_application

## Example: Adding Bessel functions

Suppose an AP needs the Bessel functions of the first and second kinds. The subtype of application_defined_function to support this can be defined as follows:

```
ENTITY bessel_function
  SUBTYPE OF (application_defined_function, generic_literal);
  SELF\application_defined_function.parameters : LIST [3:3] OF
    NUMBER;
DERIVE
  SELF\application_defined_function.explicit_domain : tuple_space
    := one_tuples_of(the_complex_numbers);
  SELF\application_defined_function.explicit_range : tuple_space
    := one_tuples_of(the_complex_numbers);
WHERE
  WR1: (SELF\application_defined_function.parameters[1] = 1) OR
       (SELF\application_defined_function.parameters[1] = 2);
  WR2: is_real(SELF\application_defined_function.parameters[2])
    AND is_real(SELF\application_defined_function.parameters[3]);
END_ENTITY;
```

# Example: Composed functions

$H(x) = h \circ g \circ f(x) = h(g(f(x)))$

is represented by entity type composed_function as

```
#2000=COMPOSED_FUNCTION((#f, #g, #h));
```

$K(x) = k(f(x), g(x), h(x))$

is represented by entity type composed_function_application as

```
#2001=COMPOSED_FUNCTION_APPLICATION( *, #k, (#f, #g, #h));
```

# Example: expression_denoted_function (1 of 2)

Consider the informal mathematical definition for the one-parameter family *F* of rotation matrices in the plane:

$$F(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

In order to recognize the expression $F(\pi/6)$ as an instance which could be used as the matrix-valued operand attribute of, say, a homogeneous_linear_function, it must first be recognized as an instance of maths_function. As constructed, the instance representing $F(\pi/6)$ is an instance of function_application. The entity type function_-application cannot be a subtype of maths_function because many of its instances do not represent mathematical functions. Instead, the nature of this particular instance of function application is recognized by wrapping it as an expression_denoted_function instance as follows:

# Example: expression_denoted_function (2 of 2)

| | |
|---|---|
| *R* | `#1600=ELEMENTARY_SPACE(.ES_REALS.);` |
| $\theta$ | `#1601=MATHS_REAL_VARIABLE(#1600,'theta');` |
| $\cos(\theta)$ | `#1602=COS_FUNCTION(#1601);` |
| $\sin(\theta)$ | `#1603=SIN_FUNCTION(#1601);` |
| $-\sin(\theta)$ | `#1604=MINUS_FUNCTION(#1603);` |
| 1D table | `#1605=LISTED_DATA(1,(#1602,#1603,#1604,#1602),#1600);` |
| matrix | `#1606=STANDARD_TABLE_FUNCTION(1,(2,2),#1605,1,.BY_ROWS.);` |
| *F* | `#1607=ABSTRACTED_EXPRESSION_FUNCTION((#1606,#1601));` |
| $\pi/6$ | `#1608=REAL_LITERAL(0.5235987);` |
| $F(\pi/6)$ | `#1609=FUNCTION_APPLICATION(*,#1607,(#1608));` |
| 2D table | `#1610=EXPRESSION_DENOTED_FUNCTION(#1609);` |

# Example: 2D Grid

**Given a list or real numbers $u_i$, i=1,…,10 and a list $v_j$, j=1,…,12, form the matrix of two-dimensional points (i.e. ordered pairs of real numbers) $[(u_i, v_j)]$.**

```
#1001=LISTED_REAL_DATA(1,*,($u_1$,$u_2$,$u_3$,$u_4$,$u_5$,$u_6$,$u_7$,$u_8$,$u_9$,$u_{10}$));
#1002=LISTED_REAL_DATA(1,*,($v_1$,$v_2$,$v_3$,$v_4$,$v_5$,$v_6$,$v_7$,$v_8$,$v_9$,$v_{10}$,$v_{11}$,$v_{12}$));
#1003=FINITE_INTEGER_INTERVAL(1,10);
#1004=FINITE_INTEGER_INTERVAL(1,12);
#1005=MATHS_INTEGER_VARIABLE(#1003,'i');
#1006=MATHS_INTEGER_VARIABLE(#1004,'j');
#1007=FUNCTION_APPLICATION(*,#1001,(#1005));
#1008=FUNCTION_APPLICATION(*,#1002,(#1006));
#1009=ELEMENTARY_FUNCTION(.EF_ENTUPLE.);
#1010=FUNCTION_APPLICATION(*,#1009,(#1007,#1008));
#1011=ABSTRACTED_EXPRESSION_FUNCTION((#1010,#1005,#1006));
```

**Instance #1011 represents the requested matrix.**

**Two instances of ENVIRONMENT and two instances of BOUND_VARIABLE_SEMANTICS not shown.**

# Example: 2D Grid notes

```
#1001= 1D table function u
#1002= 1D table function v
#1003= {n ε Z | 1 <= n <= 10}
#1004= {n ε Z | 1 <= n <= 12}
#1005= a variable i ranging over the interval 1,..,10
#1006= a variable j ranging over the interval 1,..,12
#1007= u(i), or, more commonly, u_i
#1008= v(j), or, more commonly, v_j
#1009= the elementary function which takes any number of
       arguments and creates an ordered tuple of them
#1010= expression (u_i,v_j)
#1011= 2D table function [(u_i,v_j)]
```

## Example: 2D Grid (simplified)

**Given a list or real numbers $u_i$, i=1,…,10 and a list $v_j$, j=1,…,12, form the matrix of two-dimensional points (i.e. ordered pairs of real numbers) $[(u_i, v_j)]$.**

```
#1001=LISTED_REAL_DATA(1,*,(u1,u2,u3,u4,u5,u6,u7,u8,u9,u10));
#1002=LISTED_REAL_DATA(1,*,(v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12));
#1003=FINITE_INTEGER_INTERVAL(1,10);
#1004=FINITE_INTEGER_INTERVAL(1,12);
#1009=ELEMENTARY_FUNCTION(.EF_ENTUPLE.);
#1021=SELECTION_FUNCTION((#1003,#1004),(1));
#1022=SELECTION_FUNCITON((#1003,#1004),(2));
#1023=COMPOSED_FUNCTION((#1021,#1001));
#1024=COMPOSED_FUNCTION((#1022,#1002));
#1025=COMPOSED_FUNCTION_APPLICATION(*,#1009,(#1023,#1024));
```

**Instance #1025 represents the requested matrix.**

**The two instances of ENVIRONMENT and two instances of BOUND_VARIABLE_SEMANTICS are also eliminated in this case.**

# Example: 2D Grid (simplified) notes

```
#1001= 1D table function u
#1002= 1D table function v
#1003= {n ε Z | 1 <= n <= 10}, call it [1,10]
#1004= {n ε Z | 1 <= n <= 12}, call it [1,12]
#1009= E, the elementary function which takes any number of
       arguments and creates an ordered tuple of them
```

#1021= $P_1$ projection onto 1st coordinate from [1,10] x [1,12]
#1022= $P_2$ projection onto 2nd coordinate from [1,10] x [1,12]
#1023= $u \circ P_1$
#1024= $v \circ P_2$
#1025= $E(u \circ P_1, v \circ P_2)$ = 2D table function $[(u_i,v_j)]$

## Example: 3D Grid on a Surface

Represent the grid of three-dimensional points (i.e. ordered triples of real numbers) obtained by evaluating a surface on the 2D grid: $[S(u_i,v_j)]$

```
#1101=IMPORTED_SURFACE_FUNCTION(#S,#1102);
#1102=UNIFORM_TUPLE_SPACE(#1103,2);
#1103=FINITE_REAL_INTERVAL(0.0,.CLOSED.,1.0,.CLOSED.);
#1104=REPACKAGING_FUNCTION(#1101,.RO_UNWRAP_TUPLE.,
      .RO_NOCHANGE.,0);
#1105=COMPOSED_FUNCTION((#1011,#1104));
```

#1105 is the requested 3D grid on the surface. The REPACKAGING_FUNCTION fixes the subtle discrepancy that the output of the 2D grid function is one ordered pair of reals and the input of the surface’s parametric function is two reals.

# Example: Adjoin Continuous Pressure and Temperature Distributions to a Surface

**Assume the numerical values representing Pressure and Temperature have been defined by maths_function instances P(u,v) and T(u,v), where the domains are the same as those of S(u,v) (the surface parametric function) and the ranges are real numbers.**

```
#1201=MATHS_REAL_VARIABLE(#1103,’U’);
#1202=MATHS_REAL_VARIABLE(#1103,’V’);
#1203=FUNCTION_APPLICATION(*,#P,(#1201,#1202));
#1204=FUNCTION_APPLICATION(*,#T,(#1201,#1202));
#1205=FUNCTION_APPLICATION(*,#1104,(#1201,#1202));
#1206=FUNCTION_APPLICATION(*,#1009,(#1203));
#1207=FUNCTION_APPLICATION(*,#1009,(#1204));
#1208=ELEMENTARY_FUNCTION(.EF_CONCAT_T.);
#1209=FUNCTION_APPLICATION(*,#1208,(#1205,#1206));
#1210=FUNCTION_APPLICATION(*,#1208,(#1209,#1207));
#1211=ABSTRACTED_EXPRESSION_FUNCTION((#1210,#1201,#1202));
```

**#1211 represents a function from the unit square into 5D space.**

**Two instances of ENVIRONMENT and two instances of BOUND_VARIABLE_SEMANTICS not shown.**

## Example: Adjoin Continuous Pressure and Temperature Distributions to a Surface (simplified)

**Assume the numerical values representing Pressure and Temperature have been defined by maths_function instances P(u,v) and T(u,v), where the domains are the same as those of S(u,v) (the surface parametric function) and the ranges are real numbers.**

```
#1009=ELEMENTARY_FUNCTION(.EF_ENTUPLE.);
#1104=/* the repackaged surface function S(u,v) */
#1208=ELEMENTARY_FUNCTION(.EF_CONCAT_T.);
#1221=COMPOSED_FUNCTION((#P,#1009));
#1222=COMPOSED_FUNCTION((#T,#1009));
#1223=COMPOSED_FUNCTION_APPLICATION(*,#1208,(#1104,#1221));
#1224=COMPOSED_FUNCTION_APPLICATION(*,#1208,(#1123,#1222));
```

**#1224 represents a function from the unit square into 5D space (x, y, z, P, T).**

**The two instances of ENVIRONMENT and two instances of BOUND_VARIABLE_SEMANTICS are also eliminated in this case.**

## Variation requested by Ed Stanton

**Assume the numerical values representing Pressure and Temperature have been defined throughout the space occupied by the surface S(u,v) by maths_function instances P(x,y,z) and T(x,y,z) and that their ranges are real numbers.**

```
#1009=ELEMENTARY_FUNCTION(.EF_ENTUPLE.);
#1104=/* the repackaged surface function S(u,v) */
#1208=ELEMENTARY_FUNCTION(.EF_CONCAT_T.);
#1231=COMPOSED_FUNCTION((#1104,#P,#1009));
#1232=COMPOSED_FUNCTION((#1104,#T,#1009));
#1233=COMPOSED_FUNCTION_APPLICATION(*,#1208,(#1104,#1231));
#1234=COMPOSED_FUNCTION_APPLICATION(*,#1208,(#1233,#1232));
```

**#1234 represents a function from the unit square into 5D space.**

## Example: Coefficient of Lift (1 of 2)

**A ‘property’ of a Boeing 737-700 is how a certain component of the aerodynamic coefficient of lift depends on the angle of attack and the angle of the wing flaps.  Supposing that this relationship were expressed using a b_spline_function, it would look something like the following:**

```
#1301=B_SPLINE_FUNCTION((#1302,#1303),#1304);
      /* spline bases and coefficient table */
#1302=B_SPLINE_BASIS(2,(-10.0,-10.0,-10.0,-5.0,...,35.0));
      /* degree and 23 knots for first input */
#1303=B_SPLINE_BASIS(3,(0.0,...,30.0));
      /* degree and 19 knots for second input */
#1304=STANDARD_TABLE_FUNCTION(0,(20,15),#1305,0,.BY_ROWS.);
      /* structure of table of coefficients */
#1305=LISTED_REAL_DATA(0,*,(0.134,...,0.753));
      /* 300 coefficients */
```

## Example: Coefficient of Lift (2 of 2)

**The previous slide defines the purely numerical relationship between an input of an ordered pair of reals and an output of a real. To connect this with the actual application via some variables and their "semantics" in the style of PLIB 20, the b_spline_function could be used as follows:**

```
#1306=REPACKAGING_FUNCTION(#1301,.RO_WRAP_AS_TUPLE.,
      .RO_NOCHANGE.,0);
#1307=MATHS_REAL_VARIABLE(#1309,'ANGLE_OF_ATTACK');
#1308=MATHS_REAL_VARIABLE(#1310,'FLAP_ANGLE');
#1309=FINITE_REAL_INTERVAL(-10.0,.CLOSED.,35.0,.CLOSED.);
#1310=FINITE_REAL_INTERVAL(0.0,.CLOSED.,30.0,.CLOSED.);
#1311=FUNCTION_APPLICATION(*,#1306,(#1307,#1308));
#1312=DEPENDENT_VARIABLE_DEFINITION(#1311,'COEF_LIFT_BNAS');
```

**The last entity type allows expression outputs to be documented in a manner like variables.**

# Example: Equation

**The ideal gas law states something like: PV = kT**

```
#1501=REAL_INTERVAL_FROM_MIN(0.0,.CLOSED.);
#1502=MATHS_REAL_VARIABLE(#1501,'PRESSURE');
#1503=MATHS_REAL_VARIABLE(#1501,'VOLUME');
#1504=MATHS_REAL_VARIABLE(#1501,'TEMPERATURE');
#1505=REAL_LITERAL(2.71E-5); /* not the correct number */
#1506=MULT_EXPRESSION((#1502,#1503));
#1507=MULT_EXPRESSION((#1505,#1504));
#1508=EQUALS_EXPRESSION((#1506,#1507));
```

**The units of measurement, descriptions, connections to the product, et cetera, are out of scope for this Part. They might be handled via the "variable_semantics" entities tied to each variable by the PLIB 20 schemas. Or by a maths_expression_item as discussed later in this presentation. Or by a Part 107 mechanism.**

# Example: Sparse Matrix

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 0.0 | 2.1 | 3.6 | 0.0 | 0.0 | 0.0 |
| 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | -7.1 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | -8.5 | 2.3 | 1.0 |
| 0.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 |

```
#20=BASIC_SPARSE_MATRIX(1,(5,6),(#21,#22,#23),0.0,.BY_ROWS.);
#21=LISTED_REAL_DATA(1,*,(1,3,4,5,8,9));
#22=LISTED_REAL_DATA(1,*,(  2,  3,  1,   3,   4,  5,  6,  2));
#23=LISTED_REAL_DATA(1,*,(2.1,3.6,5.0,-7.1,-8.5,2.3,1.0,2.0));
```

**Spacing added to #22 to show correspondence with #23.**

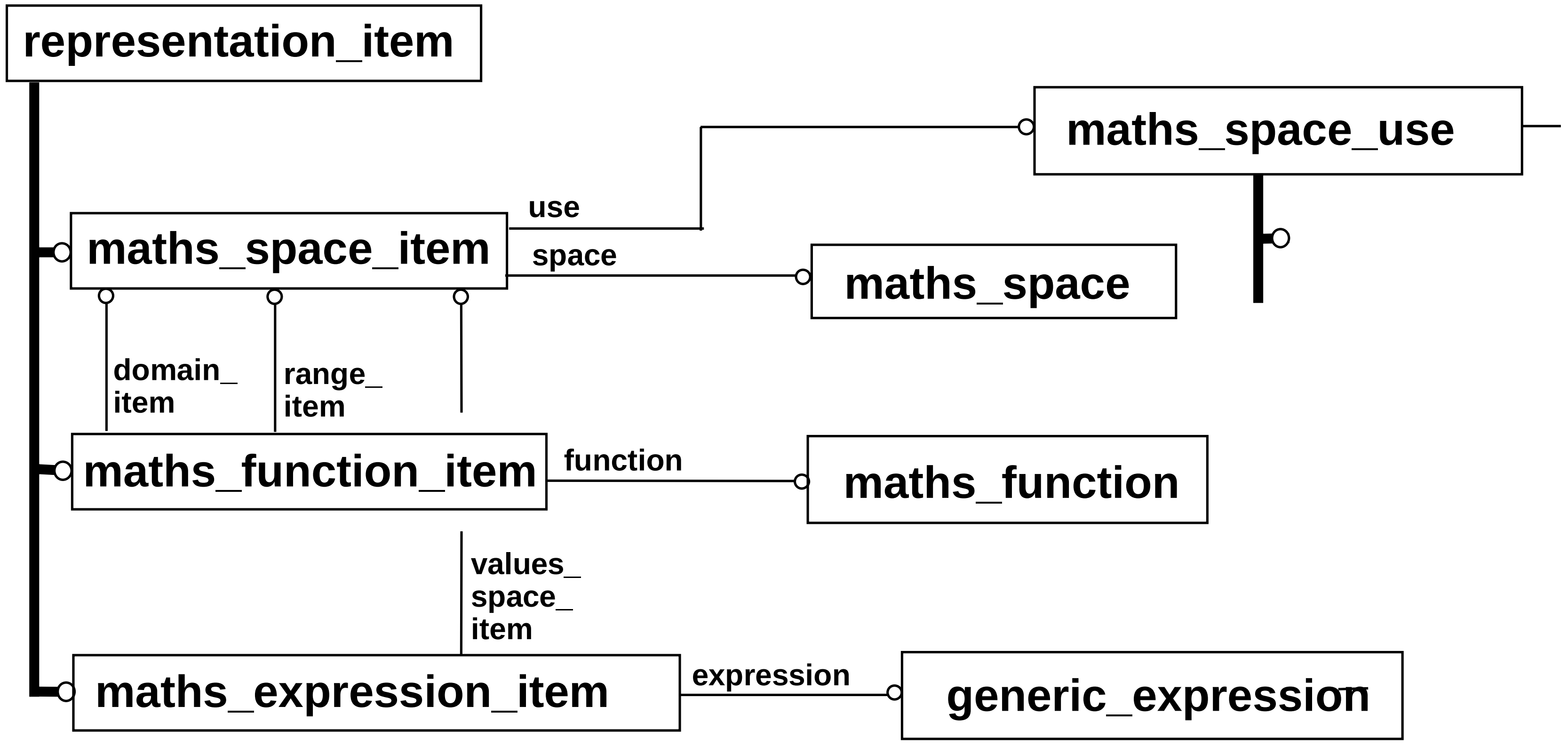
## Issue: Integration with STEP

- How is Part 50 to be used to represent complex properties of products?
- There are at least three potential answers:
  - PLIB 20 approach using entity types “variable_semantics” and “environment” to connect with expression variables.
  - Part 107 (Engineering Analysis Core Model) approach.  See David Leal.
  - Just for discussion, a fairly straightforward way to create representation items with appropriate engineering context is sketched on the next two slides.  This appears to allow use of the Part 50 entity instances to represent complex property values without any additional changes to STEP Generic Resources.

# Possible representation items from Part 50

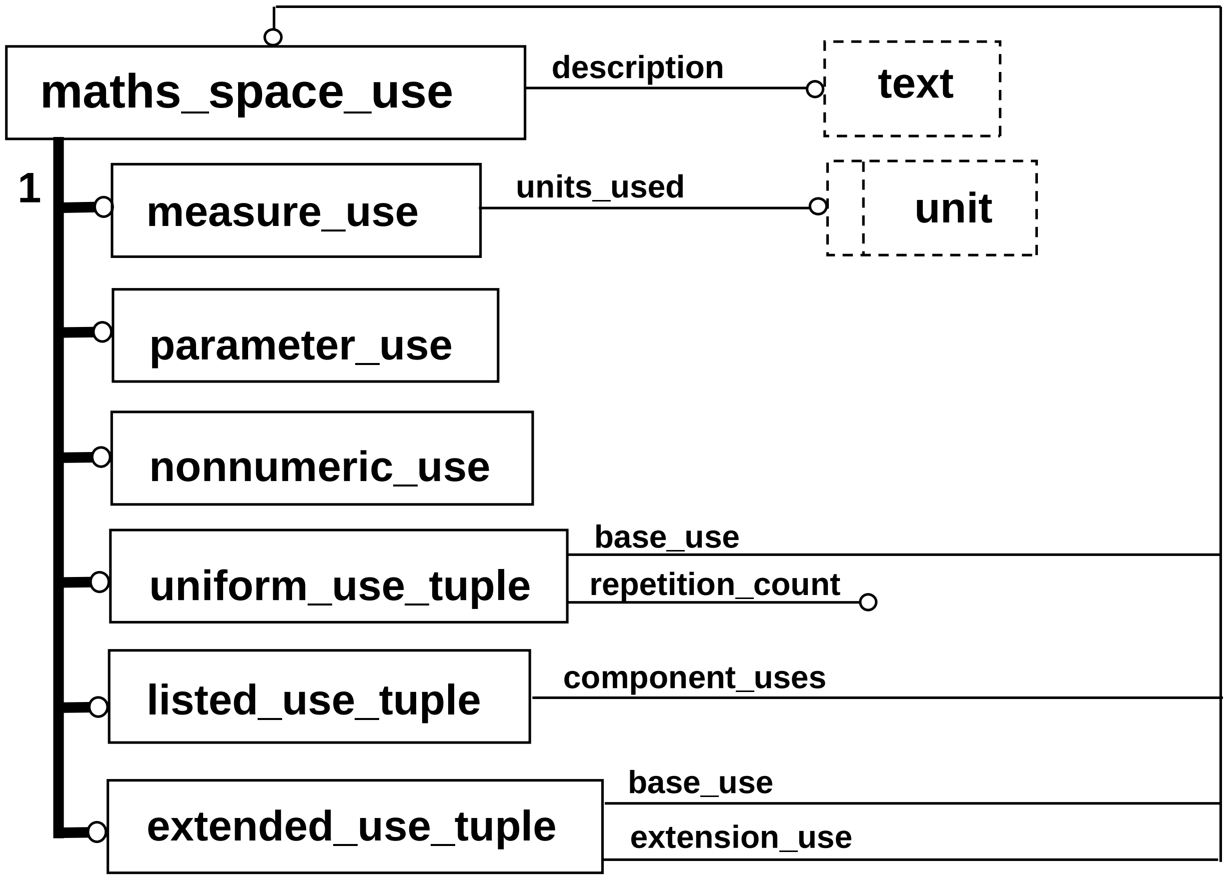
Boeing Phantom Works

Mathematics & Computing Technology



# Possible maths_space_use structure

Boeing Phantom Works
Mathematics & Computing Technology



## Issue: Logical vs. Computational Variables

**Computational Variables:**

- **Have a current value**
- **Current value changes during a computation by means of assignment operation**

**Logical Variables:**

- **Do not have a particular value, but have some space of possible values**
- **Participate in processes of 'Substitution' by which new expressions are created by means of replacement of a variable by a constant or other expression**
- **Participate in processes of 'Quantification' by which new expressions are created**

**Which kind were PLIB 20 variables intended to be?**

## Issue: Elementary function redundancy (1 of 2)

- At first sight, there appear to be two ways to represent certain elementary functions.  For concreteness, consider the sine function:
  - PLIB 20 has a sin_function entity type.
  - Part 50 has an elementary_function entity type which represents the real sine function when its attribute has the enumeration value ef_sin_r.
- In fact, however, the PLIB 20 sin_function entity requires that the sine function be applied to some operand.  That is, it efficiently represents the real-valued expression “sin (x)”, but not the sine function itself as a single mathematical object.
- Many of the constructs in Part 50 operate directly on function objects.
- Uniformity of structure and use with the other elementary functions in Part 50 which do not have PLIB 20 (or EXPRESS) counterparts favors retaining the full set in Part 50.

## Issue: Elementary function redundancy (2 of 2)

- The elementary function objects of Part 50 are the mathematical ones, which may differ subtly from the EXPRESS functions represented by PLIB 20. Consider atan2 in Part 50 versus atan in EXPRESS.
- Some redundancy is harmless and convenient. Consider that a representation for tan(x) is technically redundant since it can always be expressed as sin(x)/cos(x). Consider that the expressions x+2, 2+x, x+(3-1) are three of infinitely many ways to express essentially the same thing.

## Issue: Strange generic literals

- Many entity types in Part 50 are subtyped from PLIB 20 abstract supertype generic_literal. They do not appear to represent 'literal' data values in any ordinary sense.
- The selection of an appropriate supertype in the PLIB 20 generic expressions schema reduces to generic_literal by elimination. The entity types in question, while complex in structure, are intended to represent single mathematical objects. Consequently, they are not subtypes of generic_variable, nor can they be subtypes of unary_generic_expression, binary_generic_expression, or multiple_arity_generic_expression, all of which could have variables as operands. That leaves generic_literal. This is the supertype which contains all the instances representing constants in the language of expressions established by the ISO13584_generic_expressions_schema.

## Issue: Sine function is a constant?

- "sin" denotes a single mathematical object. Hence, its role in the language of mathematical expressions is that of a constant.
- "sin(x)" denotes an indeterminate real number. It is an expression created by applying the "sin" function to a real variable "x". In the language of mathematical expressions it is a non-elementary construct of three simple elements.
- To say that "sin" is a constant and denotes a function is not the same as saying that it is a constant function (although I may well have carelessly said exactly that late in the day at Lillehammer).

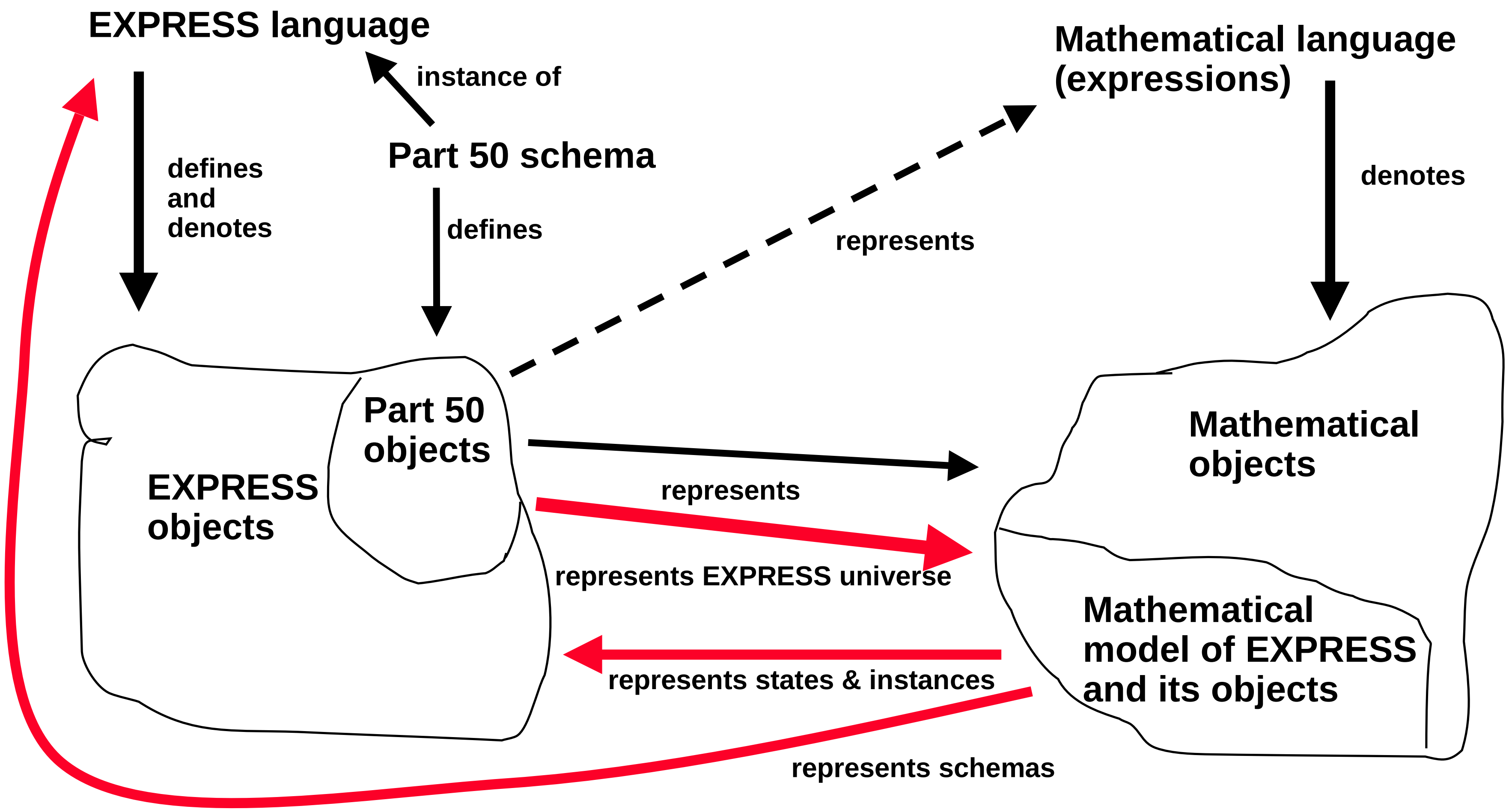
## Issue: Simple value redundancy

- There are two ways to represent all simple values and many aggregate values in Part 50. For concreteness, consider the mathematical real number three. In many Part 50 uses, this mathematical object can be represented by the EXPRESS language value “3.0”. In PLIB 20 expressions, however, it must be represented by an instance of entity type “real_literal” with attribute “the_value” containing “3.0”.
- This is a familiar and apparently unavoidable phenomenon known to all designers of object-oriented languages. It is impractical to dispense with the simple values and impossible to make them behave exactly like objects.
- The presence of this redundancy introduces extra complexity and probably some subtle inconsistencies.

# Issue: Functions operating with entity instances

- Problem: Entity instances are not mathematical objects:
  - They are created, altered and destroyed.
  - They exist only in the context of a schema and a computational environment.
  - These schemas are unknown to the Part 50 schema.
  - Since a function creating an entity instance would create a different instance each time it was called with the same inputs, such a function is not a mathematical function.
- Solution:
  - Create a mathematical model of EXPRESS schemas, computational environment states and entity instances within such environments.
  - Model computational events as mathematical functions from states to states.
  - Represent these mathematical models of schemas, states, and functions in Part 50.

# The Bigger Picture



## A Critical Distinction

Informal mathematics does not usually distinguish a space X and the space $X^1$ of one-tuples of elements of X. Part 50 finds that it must maintain that distinction.

Consider a function f which takes two real numbers as inputs. Also consider a function g which takes one ordered pair of real numbers as input. What are the domains of f and g?

The average mathematician answers $R^2$ to both and is very reluctant to admit there is a problem. The average computer programmer will see immediately that these two functions have different domains (a different “signature” in a programming language).

The answer adopted by Part 50 is that the domain of f is $R^2$ and the domain of g is $(R^2)^1$ and that these are distinct spaces.

## The Spline Knots Debate

There are two mathematically equivalent ways to describe the knots for an input of a b_spline_function - the knot sequence (with repetitions) and the breakpoints and multplicities.

### Breakpoints and multiplicities

- Two correlated list attributes
- Only occasionally used directly
- Used in geometry schema
- Claimed to be more reliable in transmission
- Causes failure of transmission in only case of different behavior
- Would maintain consistency across STEP parts
- Continuity structure more apparent

### Knot sequence

- One list attribute
- Required form for evaluation
- Proposed for Part 50
- Equally reliable in fact
- Automatically translates to nearest representable function on receiving system in the same case
- Would abandon consistency for efficiency
- Continuity structure insignificantly less apparent

## CENTRAL FOCUS

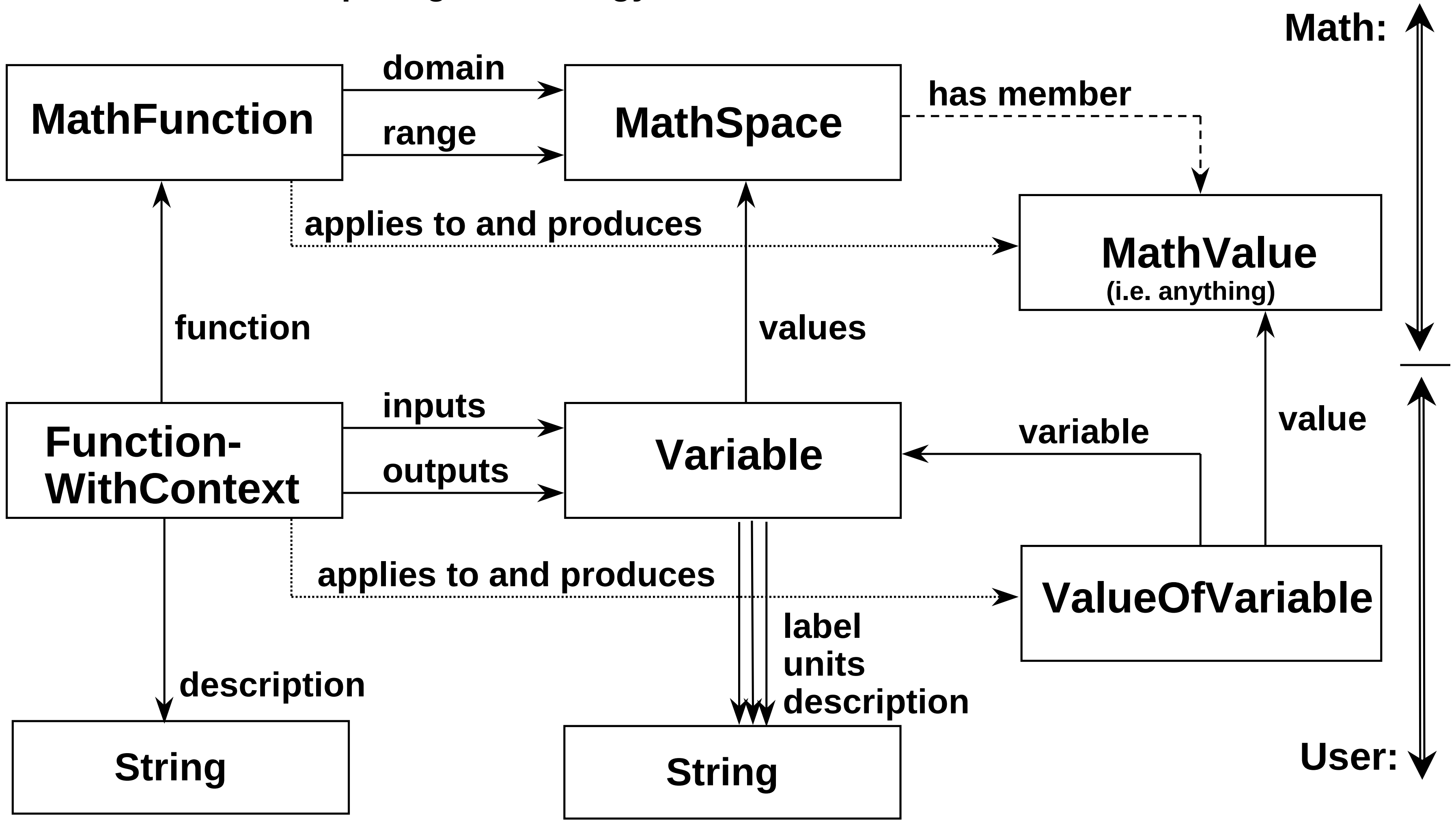
$$f : X \dashrightarrow Y$$

where f represents a mathematical function taking inputs from the mathematical space X and producing outputs in the mathematical space Y.

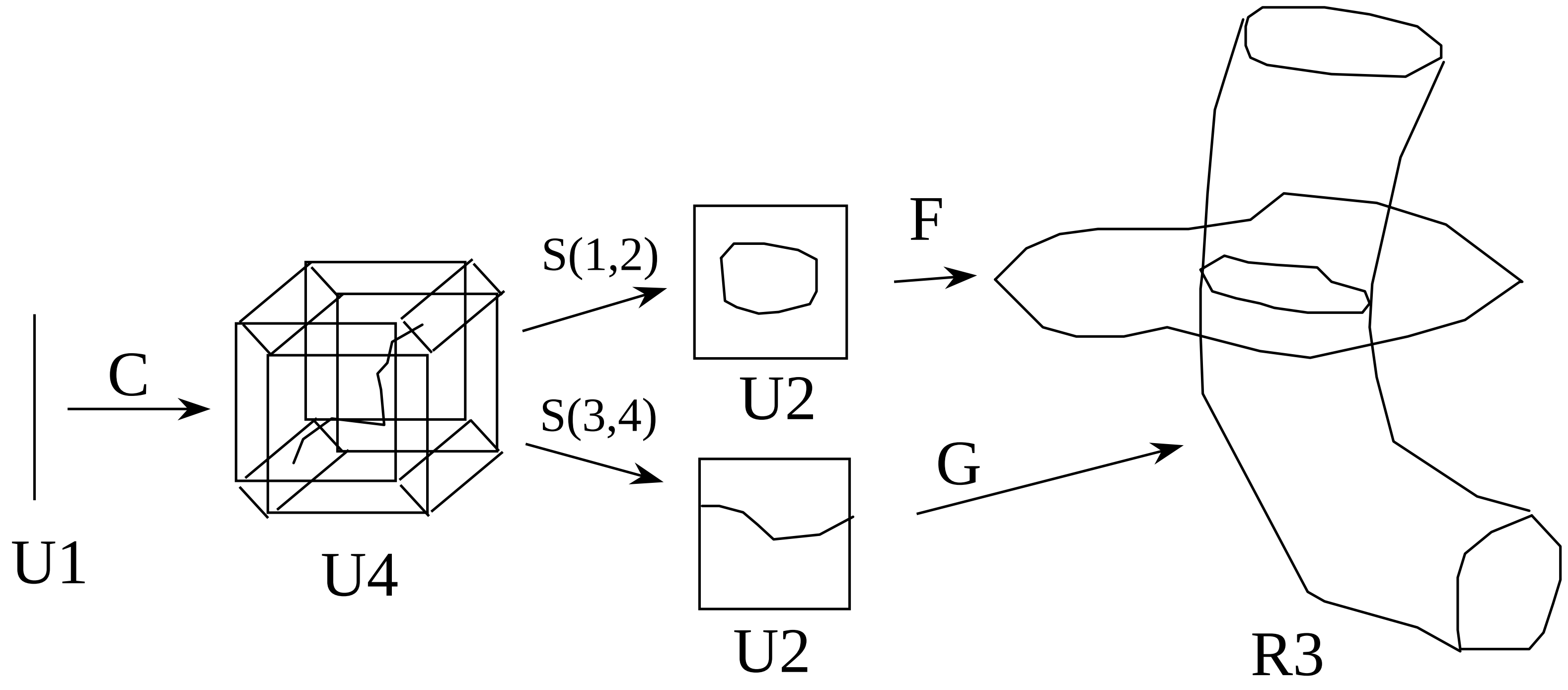
Supplementary

# Math and User Levels

Boeing Phantom Works
Mathematics & Computing Technology



# Expressing Surface Intersection


U1
C
U4
S(1,2)
S(3,4)
U2
U2
F
G
R3